hw2_Peterson_15476758

December 5, 2024

1 Homework Assignment II

Name: **Victoria Peterson**

ID: **15476758**

Course: "Performance of Networked Systems", Block II, 2024

Lecturer: Prof.dr. Rob van der Mei

Teacher Assistant: Ritul Satish

---

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.1 I : Traffic Management in IP networks

During the lecture on IP Traffic Management, two important methods to regulate incoming traffic were discussed: (1) traffic shaping, and (2) traffic policing.

1.1.1 Exercise 1

What are the main differences traffic shaping and traffic policing?

Traffic Shaping is a router functionality that aims to filter "bursty" traffic before entering a network by delaying packets that arrive in groups such that they arrive at a "peak-rate" r rather than in immediate succession.

This differs from Traffic Policing in a number of ways such that instead of "slowing-down" incoming packets, the router marks or discards packets that arrive at a rate higher than an agreed-upon mean rate r and burst tolerance b. These packets are *not* delayed like they are in Traffic Shaping and the marking rules are based on individual "contracts" made with specific classes rather than overall rate limits.

Let us now assume that the incoming – unregulated – traffic over a 50-seconds time frame looks like the left picture in Figure 1 below.

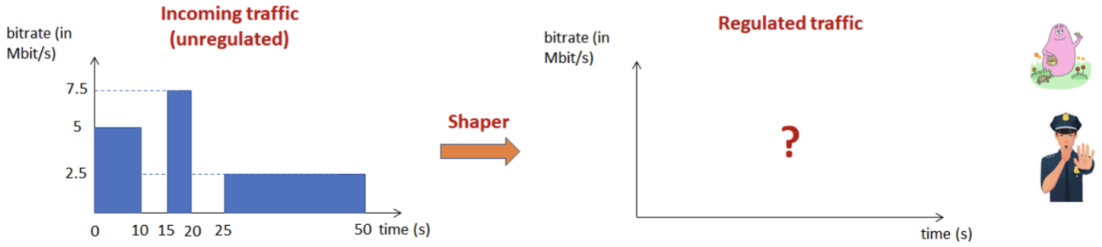


Figure 1: Unregulated traffic (left) is regulated: what does the regulated traffic look like (right)?

We first investigate the impact of traffic shaping. As discussed during the lectures, shaping functionality may introduce delay to the incoming traffic stream when the bitrate temporarily exceeds the shaping rate. That is, in terms of a Leaky Bucket (LB) implementation, the "water level" will then rise and introduce delay to incoming traffic

---

1.1.2 Exercise 2

Make a graph that shows the delay induced by the shaper (i.e., the "water level") over time. Be precise and motivate your findings

Define Traffic Rate for Timespans

[2]:
```
traffic = {
    (0, 10): 5,
    (15, 20): 7.5,
    (25, 50): 2.5
}
```

Define the Bitrate Limit for Traffic Shaping

[3]:
```
globalRate = 3.5
```

Create Array of Shaped Traffic

[4]:
```
'''
Function to create a shaped traffic array based on the rate limit
Input:
    traffic: dictionary with the traffic information
    rateLimit: rate limit to shape the traffic
'''
def trafficShaping(traffic, rateLimit):
    shapedTraffic = []
    traffic = sorted(traffic.items())

    totalTime = 0
    previousEndTime = 0

    for i in range(len(traffic)):
        time = traffic[i][0][1] - traffic[i][0][0]

        if traffic[i][1] > rateLimit:
            shapingValue = traffic[i][1]/rateLimit
        else:
            shapingValue = 1

        shapedRate = traffic[i][1]/shapingValue
        shapedTime = time*shapingValue

        initialTime = traffic[i][0][0]
        endTime =  initialTime + shapedTime

        if initialTime < totalTime:
            endTime = totalTime + shapedTime
            initialTime = totalTime
```

```
        if previousEndTime < initialTime:
            shapedTraffic.append(((previousEndTime, initialTime), 0))

        totalTime = endTime

        shapedTraffic.append(((initialTime, endTime), shapedRate))
        previousEndTime = endTime

    return shapedTraffic
'''
Output:
    shapedTraffic: shaped traffic array
'''
```

[4]: '\nOutput:\n   shapedTraffic: shaped traffic array\n'

Helper Function to find Bitrate at a Specific Time

[5]:
```
'''
Function to get the traffic bitrate at a specific time
Input:
    traffic: dictionary with the traffic information
    specificTime: specific time to get the traffic rate
'''
def getTrafficRate(traffic,  specificTime):
    for (start, end), rate in traffic.items():
        if start <= specificTime < end:
            return rate
    return 0.0
'''
Output:
    rate: traffic rate at the specific time
'''
```

[5]: '\nOutput:\n   rate: traffic rate at the specific time\n'

Function to find delay at every timestep of traffic

[6]:
```
'''
Function to calculate the traffic shaping delay
Input:
    traffic: dictionary with the traffic information
    shapedTraffic: shaped traffic array
'''
def trafficShapingDelay(traffic, shapedTraffic):
    timestepSize = 0.1
    currentDelay = 0.0
    timeArray = []
```

```
    delayArray = []

    for (start, end), rate in shapedTraffic:
        currentTime = start

        while currentTime < end:
            currentRate = getTrafficRate(traffic, currentTime)
            #print("Current Rate: ", currentRate)

            delayAdjustment = currentRate - rate
            #print("Delay Adjustment: ", delayAdjustment)

            currentDelay += delayAdjustment * timestepSize

            #print("Time: ", currentTime, "Rate: ", currentRate, "Delay: ",␣
 ↪currentDelay)

            timeArray.append(currentTime)
            delayArray.append(currentDelay)

            currentTime += timestepSize

            currentTime = np.round(currentTime, 1)

    return timeArray, delayArray
'''
Output:
    timeArray: time array
    delayArray: delay array
'''
```

[6]:
```
'\nOutput:\n    timeArray: time array\n    delayArray: delay array\n'
```

Shaped Traffic for Rate Limit of 3.5 Mbit/s

[7]:
```
shapedTraffic = trafficShaping(traffic, globalRate)
```

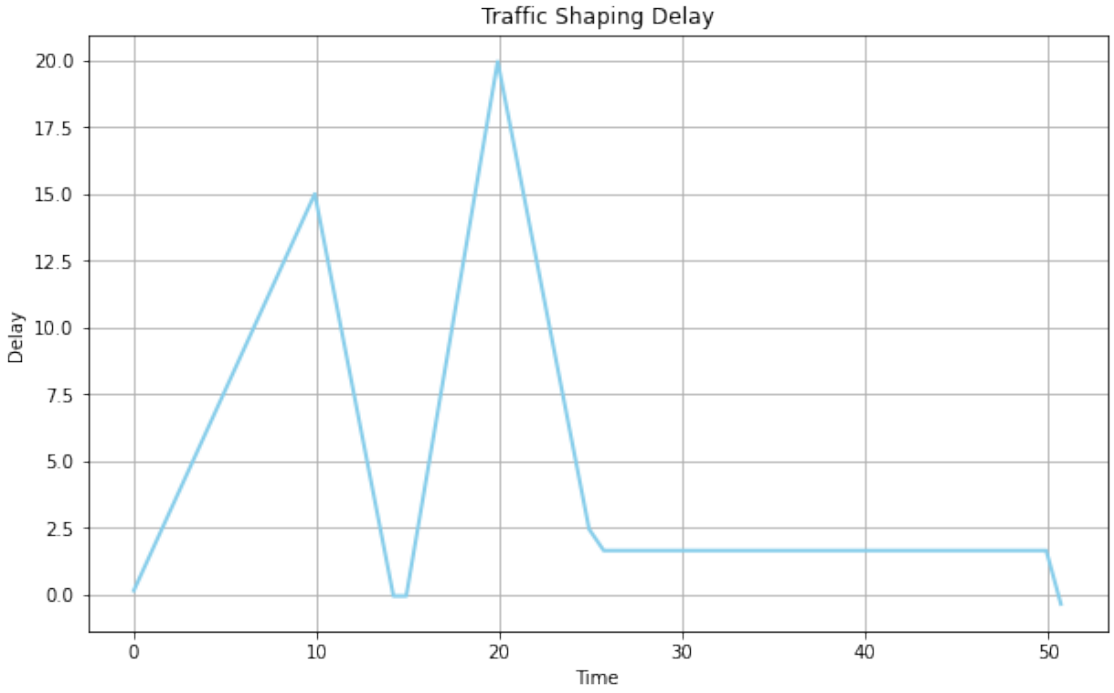
Delay from Traffic Shaping

[8]:
```
bitRateDelayTime, bitRateDelay = trafficShapingDelay(traffic, shapedTraffic)
```

Plot Accumulated Delay over time for Traffic Shaping

[9]:
```
plt.figure(figsize=(10, 6))

plt.plot(bitRateDelayTime, bitRateDelay, color='skyblue', linewidth=2)
plt.xlabel('Time')
plt.ylabel('Delay')
plt.title('Traffic Shaping Delay')
```

```
plt.grid(True)

plt.show()
```

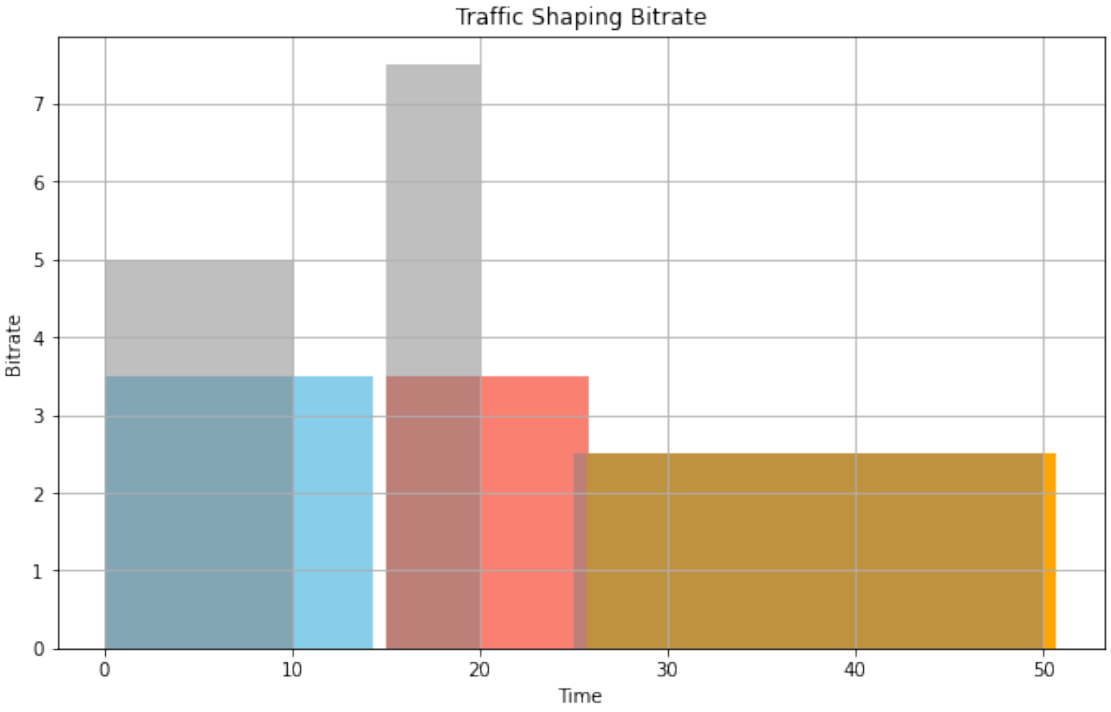


1.1.3 Exercise 3

Make a graph that plots the bitrate of the shaped traffic over time. Be precise and motivate your findings.

Plot of Shaped Traffic for Rate Limit of 3.5 **Mbit/sec**

[10]:
```
# Plots Traffic as individual bar graphs, iterating through the shaped traffic␣
 ↪and the original traffic over time steps
plt.figure(figsize=(10, 6))

colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'purple']
timestepSize = 0.1

# Plot the shaped traffic
for idx, ((start, end), rate) in enumerate(shapedTraffic):
    color = colors[idx % len(colors)]
    current_time = start
    while current_time < end:
        plt.bar(current_time, rate, width=timestepSize, color=color,␣
 ↪align='edge')
        current_time += timestepSize

for (start, end), rate in traffic.items():
    current_time = start
    while current_time < end:
        plt.bar(current_time, rate, width=timestepSize, color='gray',␣
 ↪align='edge', alpha=0.5)
        current_time += timestepSize

plt.xlabel('Time')
plt.ylabel('Bitrate')
plt.title('Traffic Shaping Bitrate')
plt.grid(True)

plt.show()
```


Traffic Shaping Bitrate
Bitrate
Time
0
1
2
3
4
5
6
7
0
10
20
30
40
50

1.1.4 Exercise 4

Answer the same question as in questions 2 and 3, but where the shaping rate is now 1.75 Mbit/s (instead of 3.5 Mbit/s)

[11]:
```
globalRate2 = 1.75
```

Shaped Traffic for Rate Limit of 1.75 Mbit/s

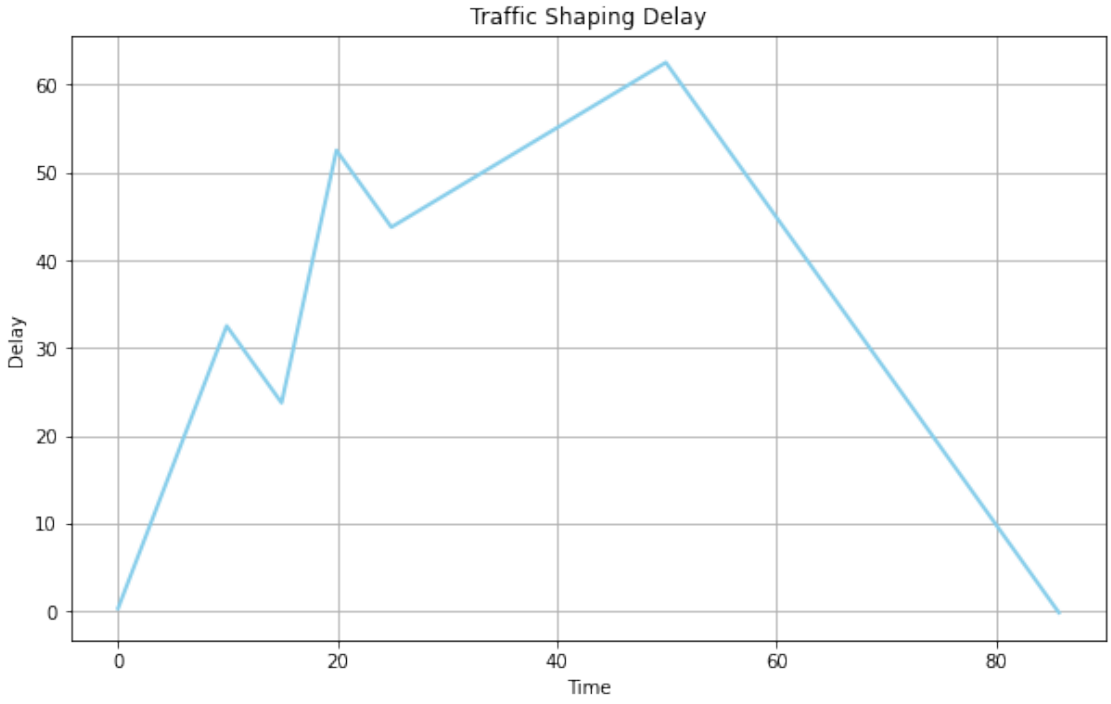
[12]:
```
shapedTraffic2 = trafficShaping(traffic, globalRate2)
```

Delay from Traffic Shaping

[13]:
```
bitRateDelayTime2, bitRateDelay2 = trafficShapingDelay(traffic, shapedTraffic2)
```

Traffic Shaping Delay Plot

[14]:
```
plt.figure(figsize=(10, 6))

plt.plot(bitRateDelayTime2, bitRateDelay2, color='skyblue', linewidth=2)
plt.xlabel('Time')
plt.ylabel('Delay')
plt.title('Traffic Shaping Delay')
plt.grid(True)

plt.show()
```



Traffic Shaping Plot

[15]:
```
# Plots Traffic as individual bar graphs, iterating through the shaped traffic␣
 ↪and the original traffic over time steps
plt.figure(figsize=(10, 6))

colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'purple']
timestepSize = 0.1

# Plot the shaped traffic
for idx, ((start, end), rate) in enumerate(shapedTraffic2):
    color = colors[idx % len(colors)]
    current_time = start
    while current_time < end:
        plt.bar(current_time, rate, width=timestepSize, color=color,␣
 ↪align='edge')
        current_time += timestepSize

for (start, end), rate in traffic.items():
    current_time = start
    while current_time < end:
        plt.bar(current_time, rate, width=timestepSize, color='gray',␣
 ↪align='edge', alpha=0.5)
        current_time += timestepSize

plt.xlabel('Time')
plt.ylabel('Bitrate')
plt.title('Traffic Shaping Bitrate')
plt.grid(True)

plt.show()
```

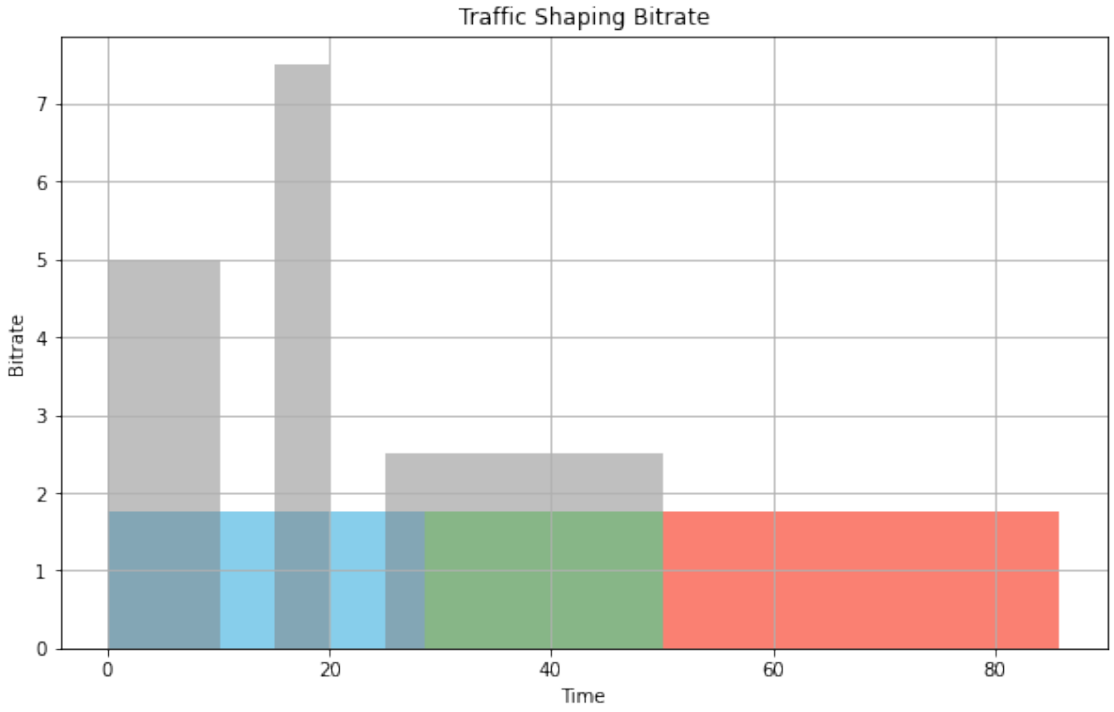

Traffic Shaping Bitrate
Bitrate
Time
0
1
2
3
4
5
6
7
0
20
40
60
80

Figure 2 below illustrates the shaping functionality for a simplistic single- burst example, and shows both the bitrate and delay induced by the shaper

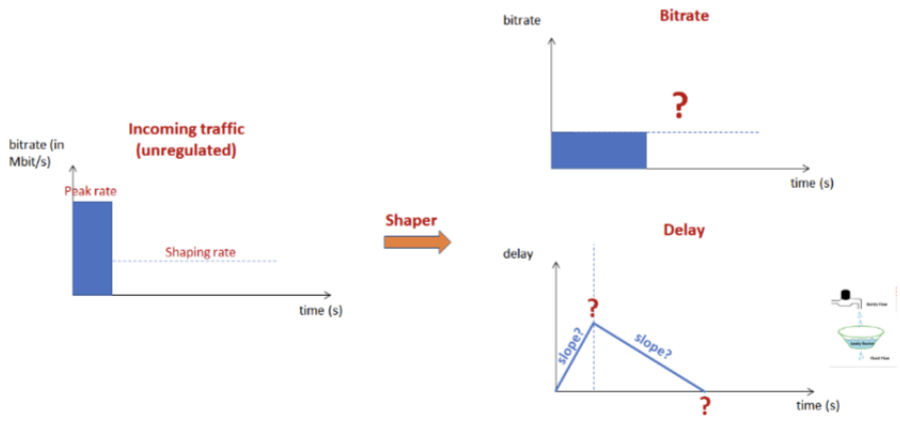


Figure 2: Illustration of the influence of shaping functionality for a single burst.

Traffic policing is a filtering mechanism that checks whether incoming traffic is conforming a Service Level Agreement (SLA), or non-conforming. Recall from the lectures that an often-used implementation of traffic policing is the LB implementation, which is characterized by two parameters: (1) the leak rate r, and (2) the burst tolerance b

Let us assume that the leak rate of r = 3.5 Mbit/s and the burst tolerance b = 1 Mbyte.

Parameters

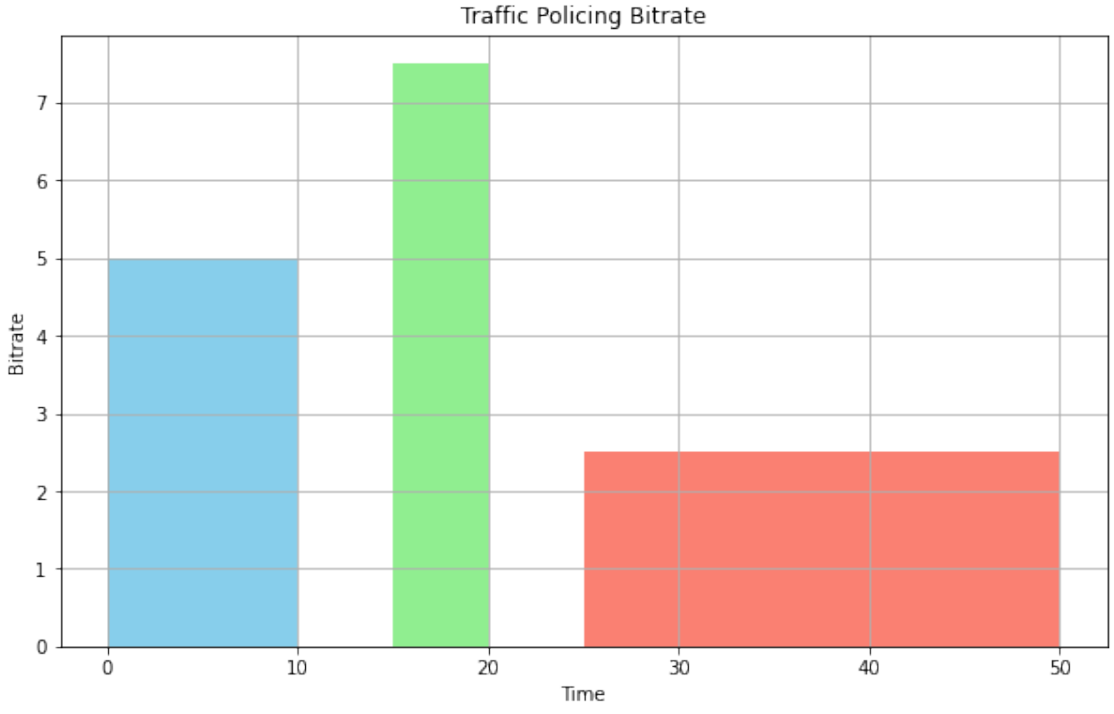
[28]:
```
leakRate = 3.5

burstTolerance = 8

timestepSize = 0.1
```

1.1.5 Exercise 5

Make a graph that plots the bitrate of the policed traffic over time

Traffic bitrate remains the same as the policed traffic is not delayed, only marked when in excess of the contract

[29]:
```
plt.figure(figsize=(10, 6))

colors = ['skyblue', 'lightgreen', 'salmon', 'orange', 'purple']

for idx, ((start, end), rate) in enumerate(traffic.items()):
    print(start, end, rate)
    color = colors[idx % len(colors)]
    for i in range(int((end - start))):
        plt.bar(int(start + i), rate, width=1, color=color, align='edge')

plt.xlabel('Time')
plt.ylabel('Bitrate')
plt.title('Traffic Policing Bitrate')
plt.grid(True)

plt.show()
```



Traffic policing is a filtering mechanism that checks whether incoming traffic is conforming a Service Level Agreement (SLA), or non-conforming

---

1.1.6 Exercise 6

Make a two-colored graph that shows the parts of in coming traffic that are conforming the SLA in blue, and the non-conforming traffic in red, over time.

Function to Perform Traffic Policing

[30]:
```
"""
Function to perform traffic policing with marking
Input:
    traffic: dictionary with the traffic information
    leakRate: leak rate
    burstTolerance: burst tolerance
    timestepSize: timestep size
"""
def trafficPolicing(traffic, leakRate, burstTolerance, timestepSize):
    timeArray = []
    unmarkedTraffic = []
    trafficRate = []
    remainingTokensarray = []
    waterlevelArray = []

    currentTime = 0.0
    totalWaterLevel = 0.0
    traffic = sorted(traffic.items())

    for (start, end), rate in traffic:
        remainingTokens = burstTolerance

        # Handle time before the start of the current traffic interval
        while currentTime < start:
            timeArray.append(currentTime)
            trafficRate.append(0.0)
            unmarkedTraffic.append(0.0)
            remainingTokensarray.append(remainingTokens)
            waterlevelArray.append(totalWaterLevel)

            currentTime = np.round(currentTime + timestepSize, 1)
            remainingTokens = min(burstTolerance, remainingTokens + leakRate *␣
 ↪timestepSize)

            totalWaterLevel = max(0.0, totalWaterLevel - leakRate * timestepSize)

        # Handle time during the current traffic interval
```

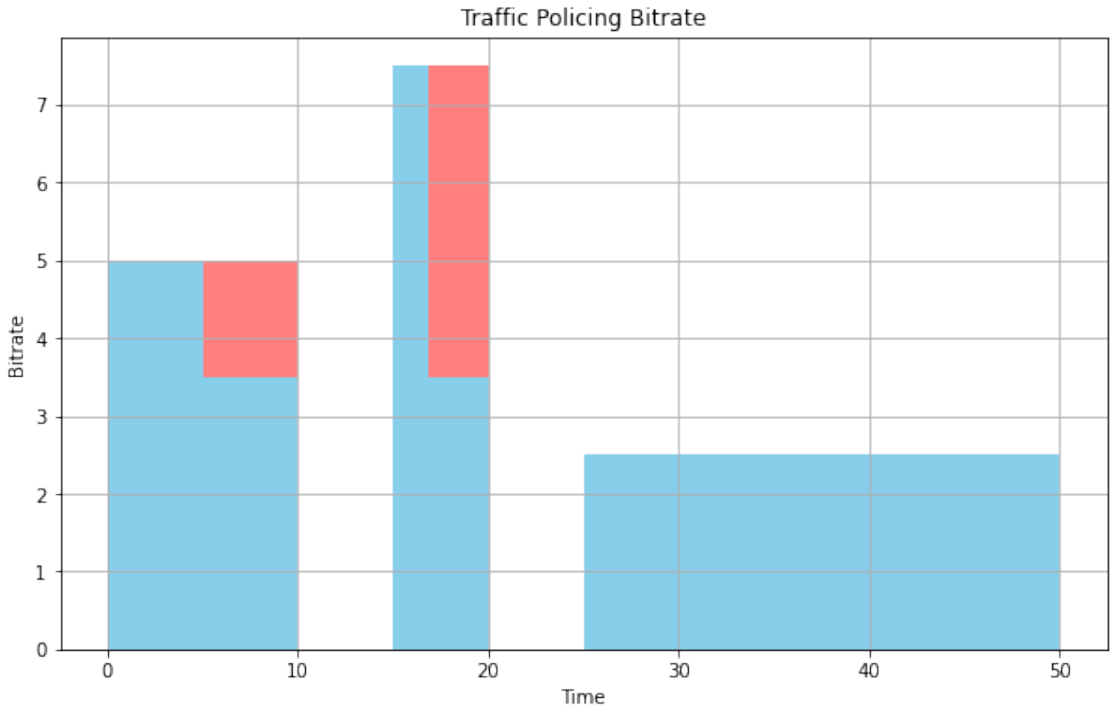
```
        while currentTime < end:
            timeArray.append(currentTime)
            trafficRate.append(rate)
            remainingTokensarray.append(remainingTokens)
            waterlevelArray.append(totalWaterLevel)

            currentTraffic = rate * timestepSize

            # Check if all of the current traffic can be sent
            if currentTraffic > remainingTokens:
                unmarkedTraffic.append(leakRate)
                remainingTokens -= leakRate * timestepSize

                totalWaterLevel += leakRate * timestepSize
            else:
                unmarkedTraffic.append(rate)
                remainingTokens -= currentTraffic

                totalWaterLevel += currentTraffic

            remainingTokens = min(burstTolerance, remainingTokens + leakRate *␣
 ↪timestepSize)
            totalWaterLevel = max(0.0, totalWaterLevel - leakRate * timestepSize)

            currentTime = np.round(currentTime + timestepSize, 1)

    return timeArray, unmarkedTraffic, trafficRate, remainingTokensarray,␣
 ↪waterlevelArray
"""
Output:
    timeArray: time array
    unmarkedTraffic: unmarked traffic array
    trafficRate: traffic rate array
    remainingTokensarray: remaining tokens array
"""
```

[30]:
```
'\nOutput:\n    timeArray: time array\n    unmarkedTraffic: unmarked traffic
array\n    trafficRate: traffic rate array\n    remainingTokensarray: remaining
tokens array\n'
```

[31]:
```
policingTime, policingUnmarked, policingTrafficRate, tokensArray,␣
 ↪waterlevelArray = trafficPolicing(traffic, leakRate, burstTolerance,␣
 ↪timestepSize)
```

Plot the Marked Traffic from Traffic Policing

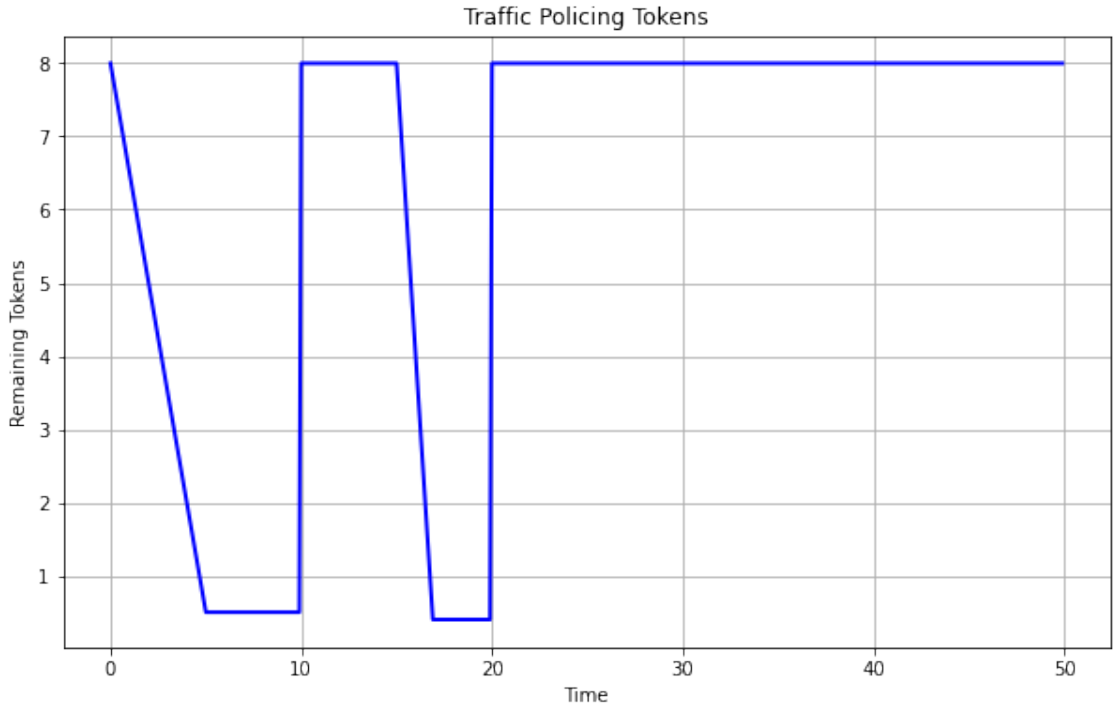
[32]:
```
plt.figure(figsize=(10, 6))

plt.bar(policingTime, policingTrafficRate, width=timestepSize, color='red',␣
 ↪align='edge', alpha=0.5)
plt.bar(policingTime, policingUnmarked, width=timestepSize, color='skyblue',␣
 ↪align='edge')

plt.xlabel('Time')
plt.ylabel('Bitrate')
plt.title('Traffic Policing Bitrate')
plt.grid(True)

plt.show()
```



Plot the Number of Tokens available over time

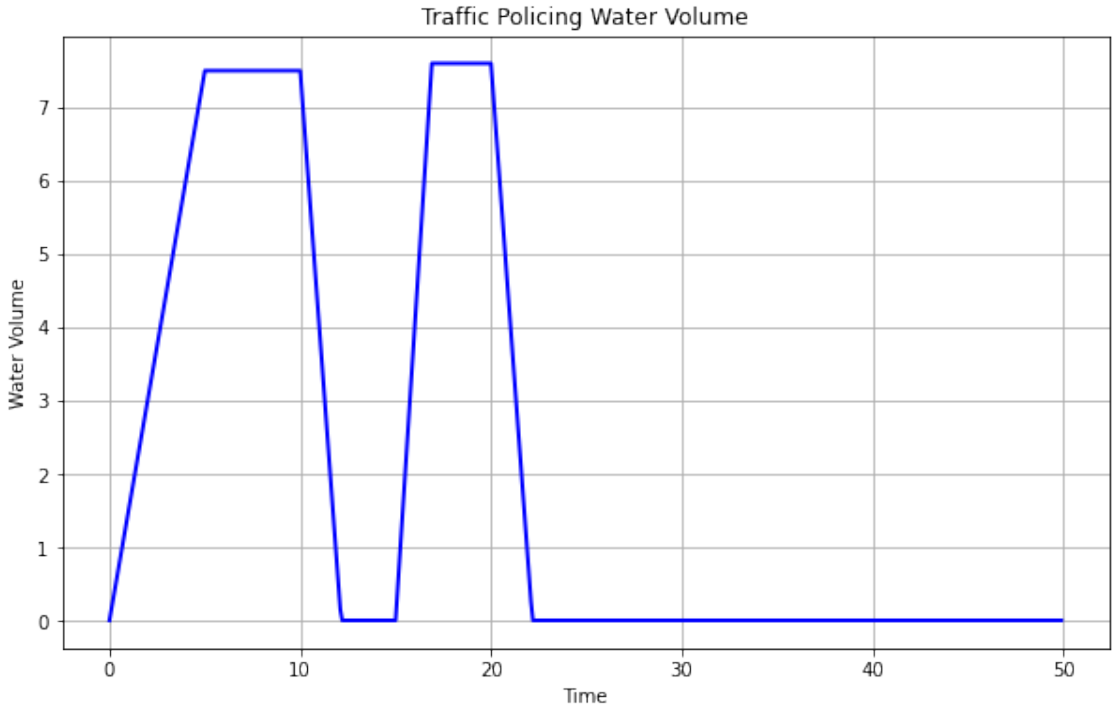
[33]:
```
plt.figure(figsize=(10, 6))

plt.plot(policingTime, tokensArray, color='blue', linewidth=2)

plt.xlabel('Time')
plt.ylabel('Remaining Tokens')
plt.title('Traffic Policing Tokens')
plt.grid(True)
```

```
plt.show()
```



Plot the "Water Volume" of the "Bucket"

[34]:
```
plt.figure(figsize=(10, 6))

plt.plot(policingTime, waterlevelArray, color='blue', linewidth=2)
plt.xlabel('Time')
plt.ylabel('Water Volume')
plt.title('Traffic Policing Water Volume')
plt.grid(True)

plt.show()
```


Traffic Policing Water Volume
Water Volume
Time
0
1
2
3
4
5
6
7
0
10
20
30
40
50

1.1.7 Exercise 7

Answer the same questions as in questions 5 and 6, but where the leak rate and shaping rate are now 1.75 Mbit/s (instead of 3.5 Mbit/s).

Parameters

[35]:
```
shapingRate = 1.75

leakRate2 = 1.75

burstTolerance2 = 8

timestepSize = 0.1
```

Shape Traffic

[36]:
```
shapedTraffic3 = trafficShaping(traffic, shapingRate)
```

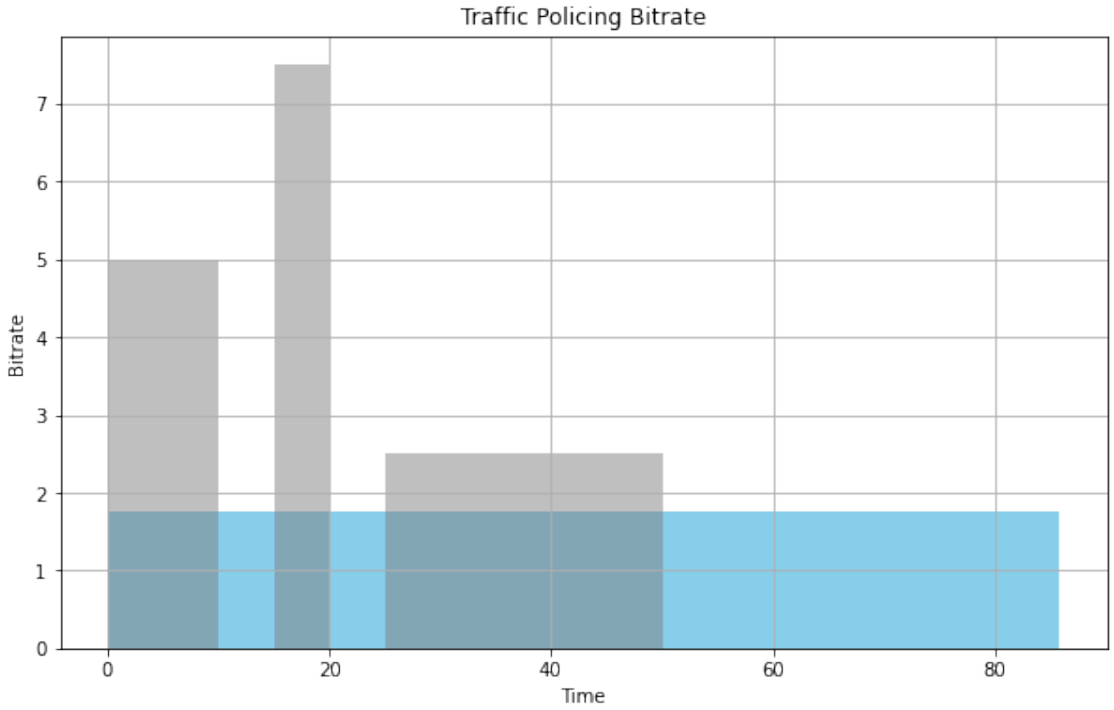
[37]:
```
# Convert Shaped traffic to dictionary to then be policed
dictShapedTraffic3 = {timespan: rate for timespan, rate in shapedTraffic3}
```

Police Traffic

[38]:
```
policingTime2, policingUnmarked2, policingTrafficRate2, tokensArray2,␣
 ↪waterlevelArray2 = trafficPolicing(dictShapedTraffic3, leakRate2,␣
 ↪burstTolerance2, timestepSize)
```

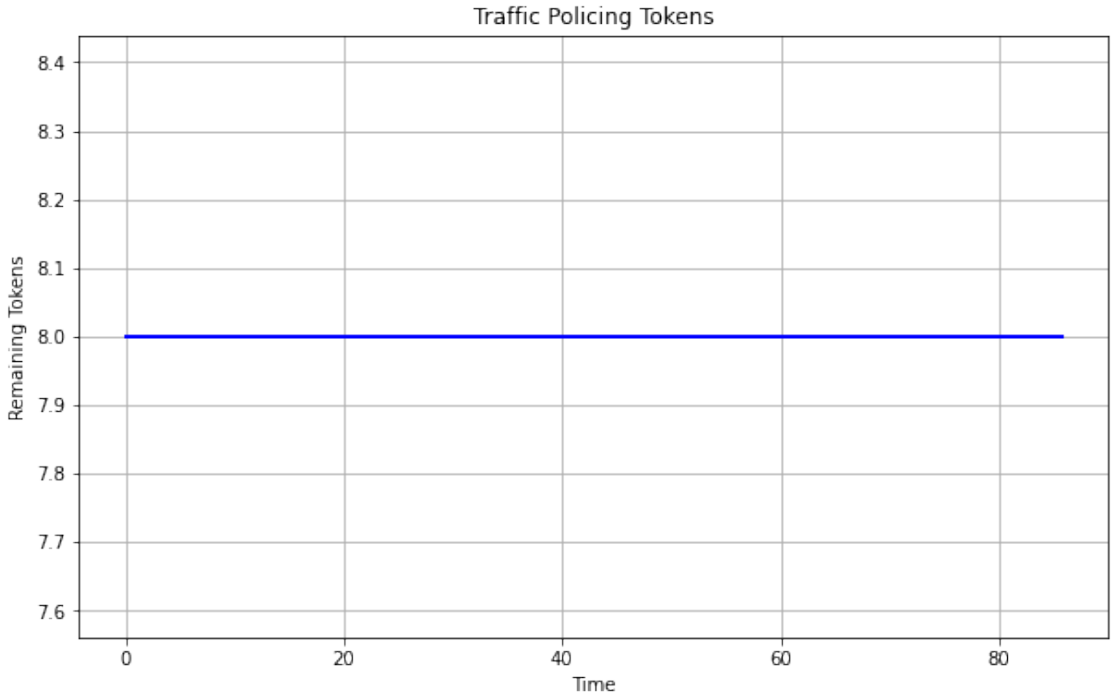
Plot the Marked Traffic from Traffic Policing

[39]:
```
plt.figure(figsize=(10, 6))

plt.bar(policingTime2, policingTrafficRate2, width=timestepSize, color='red',␣
 ↪align='edge', alpha=0.5)
plt.bar(policingTime2, policingUnmarked2, width=timestepSize, color='skyblue',␣
 ↪align='edge')

for (start, end), rate in traffic.items():
    for i in range(int((end - start))):
        plt.bar(int(start + i), rate, width=1, color='gray', align='edge',␣
 ↪alpha=0.5)

plt.xlabel('Time')
plt.ylabel('Bitrate')
plt.title('Traffic Policing Bitrate')
plt.grid(True)

plt.show()
```



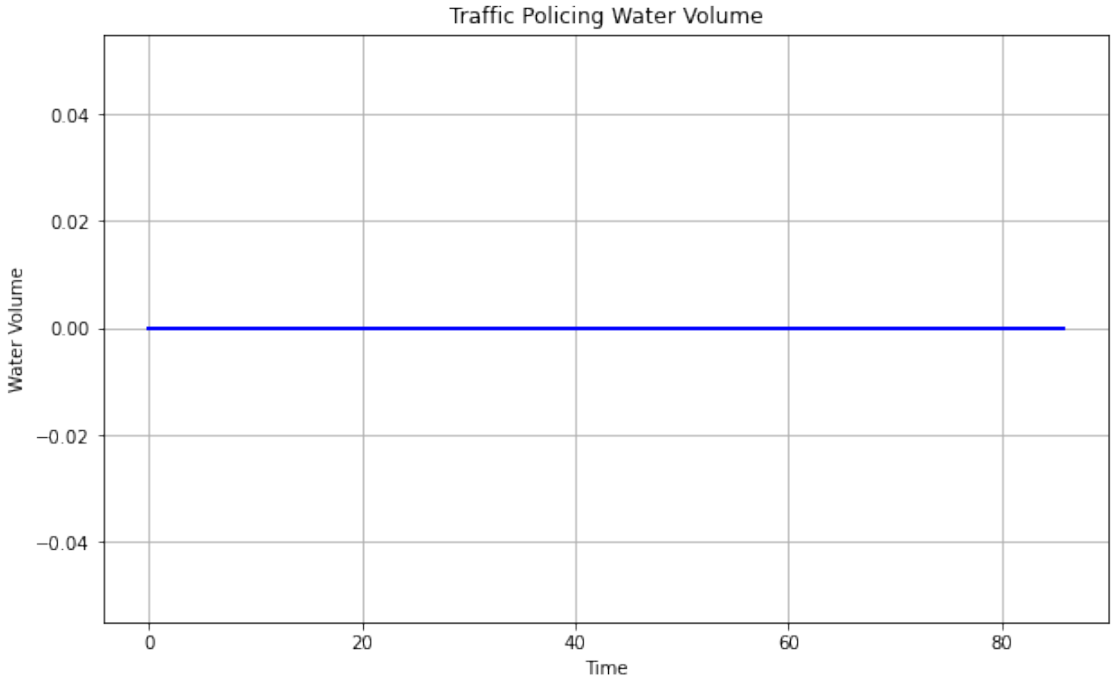
Plot the Number of Tokens available over time

[40]:
```
plt.figure(figsize=(10, 6))

plt.plot(policingTime2, tokensArray2, color='blue', linewidth=2)

plt.xlabel('Time')
plt.ylabel('Remaining Tokens')
plt.title('Traffic Policing Tokens')
plt.grid(True)

plt.show()
```



Plot the "Water Volume" of the "Bucket"

[41]:
```
plt.figure(figsize=(10, 6))

plt.plot(policingTime2, waterlevelArray2, color='blue', linewidth=2)
plt.xlabel('Time')
plt.ylabel('Water Volume')
plt.title('Traffic Policing Water Volume')
plt.grid(True)

plt.show()
```


Traffic Policing Water Volume
0.04
0.02
0.00
−0.02
−0.04
Water Volume
0
20
40
60
80
Time

1.2 II : Performance of TCP-based networks

During the lectures, we have discussed how the TCP protocol works, including the evolution of the congestion window, acknowledgements, TCP Slow Start, etc.

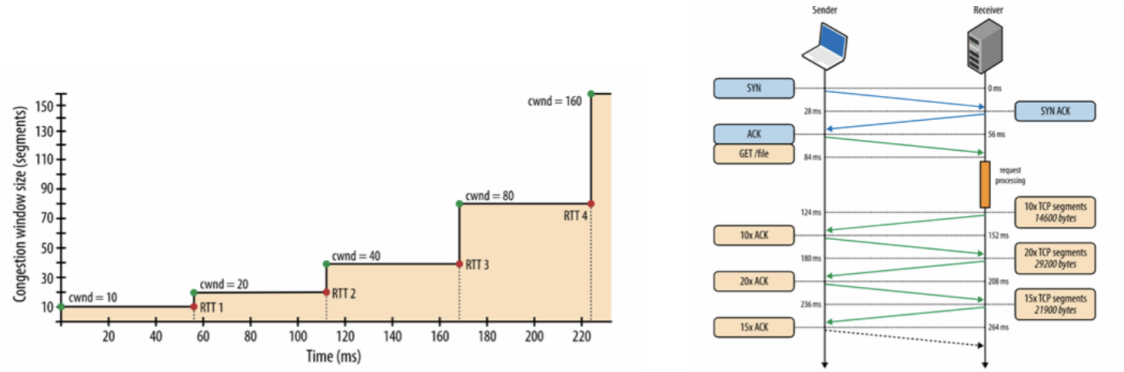


Figure 3: Illustration of the evolution of the congestion window in TCP Slow Start (left) and the acknowledgement process (right) during a TCP file transfer (from the book by Grigorik).

1.2.1 Exercise 8

What is the main downside of TCP Slow Start?

TCP Slow Start as a Congestion Control mechanism prevents network congestion by gradually increasing the amount of data sent over the network over time, but this process has some downsides. The greatest downside is one that stems from the very structure of the system in which the slow start begins with a small congestion window, meaning the traffic cannot maximize the usage of available bandwidth immediately. This issue may be mitigated with a larger initial congestion window size or the implementation of TCP Fast open to allow data to be sent simultaneously with the handshake phase.

Suppose that we want to transfer a file from Amsterdam and Atlanta, GA, over a long-distance transatlantic TCP-connection with the following characteristics: RTT = 60ms, receive CW size = 256KB, bandwidth = 200 Mbit/s, MSS = 1460 Bytes, and where the request processing time to generate response = 50ms.

Unit-Converted Parameters

Round Trip time = 0.06 seconds

Congestion Window = 256 KB

Bandwidth = 24414.0625 KB/sec

Maximum Segment Size = 1.426 KB

Processing Time = 0.05 seconds

1.2.2 Exercise 9

For file sizes 15KBytes, 25KBytes and 40KBytes, calculate the transfer time when the initial CW size is 1 segment, assuming that the transfer starts off with a three-way handshake, and is completely processed in Slow Start

Transfer Time Calculations

Three-way handshake = 1.5 Roundtrips = 0.09 seconds

With TCP Fast Open, the client sends the `get` request along with the ACK, thus incuring only 1 RTT

Slow Start: - 1 RTT : 1 MSS - 2 RTT : 2 MSS + 1 MSS = 3 MSS - 3 RTT : 4 MSS + 2 MSS + 1 MSS = 7 MSS - 4 RTT : 8 MSS + 4 MSS + 2 MSS + 1 MSS = 15 MSS - 5 RTT : 16 MSS + 8 MSS + 4 MSS + 2 MSS + 1 MSS = 31 MSS

15 KBytes

Number of Segments = File Size/MSS = $15/1.426 \approx 11$

11 Segments requires 4 roundtrips given an initial conditon window of 1 MSS

The time for 4 roundtrips is $0.06 \times 4 = 0.24$

The total transfer time is the sum of... - Three Way Handshake = 0.09 - Request Processing Time = 0.05 - Slow Start Time = 0.24

$$\text{Transfer Time } = 0.09 + 0.05 + 0.24 = 0.38 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.06 + 0.05 + 0.24 = 0.35 \text{ seconds}$$

25 KBytes

Number of Segments = File Size/MSS = $25/1.426 \approx 18$

18 Segments requires 5 roundtrips given an initial conditon window of 1 MSS

The time for 5 roundtrips is $0.06 \times 5 = 0.3$

The total transfer time is the sum of... - Three Way Handshake = 0.09 - Request Processing Time = 0.05 - Slow Start Time = 0.3

$$\text{Transfer Time } = 0.09 + 0.05 + 0.3 = 0.44 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.06 + 0.05 + 0.3 = 0.41 \text{ seconds}$$

40 KBytes

Number of Segments = File Size/MSS = $40/1.426 \approx 29$

29 Segments requires 5 roundtrips given an initial conditon window of 1 MSS

The time for 5 roundtrips is $0.06 \times 5 = 0.3$

The total transfer time is the sum of... - Three Way Handshake = 0.09 - Request Processing Time = 0.05 - Slow Start Time = 0.3

$$\text{Transfer Time } = 0.09 + 0.05 + 0.3 = 0.44 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.06 + 0.05 + 0.3 = 0.41 \text{ seconds}$$

1.2.3 Exercise 10

Answer the same questions (as in question 8), but now assuming that the initial CW size of the TCP connection is two segments

Transfer Time Calculations

Slow Start: - 1 RTT : 2 MSS - 2 RTT : 4 MSS + 2 MSS = 6 MSS - 3 RTT : 8 MSS + 4 MSS + 2 MSS = 14 MSS - 4 RTT : 16 MSS + 8 MSS + 4 MSS + 2 MSS = 30 MSS

15 KBytes

Number of Segments = File Size/MSS = $15/1.426 \approx 11$

11 Segments requires 3 roundtrips given an initial conditon window of 2 MSS

The time for 3 roundtrips is $0.06 \times 3 = 0.18$

The total transfer time is the sum of... - Three Way Handshake = 0.09 - Request Processing Time = 0.05 - Slow Start Time = 0.18

$$\text{Transfer Time } = 0.09 + 0.05 + 0.18 = 0.32 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.06 + 0.05 + 0.18 = 0.29 \text{ seconds}$$

25 KBytes

Number of Segments = File Size/MSS = $25/1.426 \approx 18$

18 Segments requires 5 roundtrips given an initial conditon window of 2 MSS

The time for 4 roundtrips is $0.06 \times 4 = 0.24$

The total transfer time is the sum of... - Three Way Handshake = 0.09 - Request Processing Time = 0.05 - Slow Start Time = 0.24

$$\text{Transfer Time } = 0.09 + 0.05 + 0.24 = 0.38 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.06 + 0.05 + 0.24 = 0.35 \text{ seconds}$$

40 KBytes

Number of Segments = File Size/MSS = $40/1.426 \approx 29$

29 Segments requires 5 roundtrips given an initial conditon window of 2 MSS

The time for 4 roundtrips is $0.06 \times 4 = 0.24$

The total transfer time is the sum of... - Three Way Handshake = 0.09 - Request Processing Time = 0.05 - Slow Start Time = 0.24

$$\text{Transfer Time } = 0.09 + 0.05 + 0.24 = 0.38 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.06 + 0.05 + 0.24 = 0.35 \text{ seconds}$$

1.2.4 Exercise 11

Answer the same question (as in question 9), but now assuming that the RTT is doubled to 120ms (instead of 60 ms).

Transfer Time Calculations

Converting all units to seconds and Kilobytes...

Round Trip time = 0.12 seconds

Congestion Window = 256 KB

Bandwidth = 24414.0625 KB/sec

Maximum Segment Size = 1.426 KB

Processing Time = 0.05 seconds

Three-way handshake = 1.5 Roundtrips = 0.18 seconds

With TCP Fast Open, the client sends the `get` request along with the ACK, thus incuring only 1 RTT

Slow Start: - 1 RTT : 1 MSS - 2 RTT : 2 MSS + 1 MSS = 3 MSS - 3 RTT : 4 MSS + 2 MSS + 1 MSS = 7 MSS - 4 RTT : 8 MSS + 4 MSS + 2 MSS + 1 MSS = 15 MSS - 5 RTT : 16 MSS + 8 MSS + 4 MSS + 2 MSS + 1 MSS = 31 MSS

15 KBytes

Number of Segments = File Size/MSS = $15/1.426 \approx 11$

11 Segments requires 4 roundtrips given an initial conditon window of 1 MSS

The time for 4 roundtrips is $0.12 \times 4 = 0.48$

The total transfer time is the sum of... - Three Way Handshake = 0.18 - Request Processing Time = 0.05 - Slow Start Time = 0.48

$$\text{Transfer Time } = 0.18 + 0.05 + 0.48 = 0.71 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.06$ seconds

$$\text{Transfer Time } = 0.12 + 0.05 + 0.48 = 0.65 \text{ seconds}$$

25 KBytes

Number of Segments = File Size/MSS = $25/1.426 \approx 18$

18 Segments requires 5 roundtrips given an initial conditon window of 1 MSS

The time for 5 roundtrips is $0.12 \times 5 = 0.60$

The total transfer time is the sum of... - Three Way Handshake = 0.18 - Request Processing Time = 0.05 - Slow Start Time = 0.6

$$\text{Transfer Time } = 0.18 + 0.05 + 0.6 = 0.83 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.12$ seconds

$$\text{Transfer Time } = 0.12 + 0.05 + 0.6 = 0.77 \text{ seconds}$$

40 KBytes

Number of Segments = File Size/MSS = $40/1.426 \approx 29$

29 Segments requires 5 roundtrips given an initial conditon window of 1 MSS

The time for 5 roundtrips is $0.12 \times 5 = 0.60$

The total transfer time is the sum of... - Three Way Handshake = 0.18 - Request Processing Time = 0.05 - Slow Start Time = 0.6

$$\text{Transfer Time } = 0.18 + 0.05 + 0.6 = 0.83 \text{ seconds}$$

With TCP Fast Open, the initial handshake is reduced to 1 RTT $= 0.12$ seconds

$$\text{Transfer Time } = 0.12 + 0.05 + 0.6 = 0.77 \text{ seconds}$$

1.2.5 Exercise 12

Use insights obtained in questions 8, 9 and 10 to give a formula that expresses the transfer time (in Slow Start, and including the three-way handshake) in terms of the file size, the RTT, the MSS and the server processing times (both the receive CW size and the bandwidth are assumed to be very large such that they do not play a role).

Parameters

Known

R : Roundtrip Time

M : Maximum Segment Size

P : Processing Time

F : File Size

C : Initial Congestion Window

Unknown

S : Number of segments required for the file F

N : Number of Roundtrips required for files transfer

H_{SS} : Three-way Handshake Slow Start

H_{FO} : TCP Fast-Open

T_{SS} : Total Transfer Time with Three-Way Handshake

T_{FO} : Total Transfer Time with TCP Fast Open

Three-Way Handshake Time

$$H_{SS} = 1.5 \cdot R$$

$$H_{FO} = 1.0 \cdot R$$

Number of Segments required

Rounding to the next highest whole number...

$$S = \frac{F}{M}$$

Number of Round Trips required for S Segments of M size

Find N roundtrips such that...

$$\sum_{n=1}^{N} 2^{n-1} \cdot C \geq S$$

Total Transfer time for file of size F

$$T_{SS} = H_{SS} + P + (N \cdot R)$$

$$T_{FO} = H_{FO} + P + (N \cdot R)$$

Approximating the Number of Round Trips N required for S Segments

By approximating the equation for the number of roundtrips required for S segments of M size, the formula may be adjusted to be inserted into the Total Transfer Time Equation.

$$\sum_{n=1}^{N} 2^{n-1} \cdot C \geq S$$

$$C \cdot \sum_{n=1}^{N} 2^{n-1} \geq S$$

$\sum_{n=1}^{N} 2^{n-1}$ is a geometric series where $a_1 = 1$ such that...

$$\frac{2^N - 1}{2 - 1} = 2^N - 1$$

$$(2^N - 1) \cdot C \geq S$$

Where $C > 0$...

$$2^N - 1 \geq \frac{S}{C}$$

$$2^N \geq \frac{S}{C} + 1$$

$$N \geq \log_2(\frac{S}{C} + 1)$$

This gives N roundtrips to be rounded up to the next highest whole number, but to better fit into the Total Transfer Time Equation, this may be approximated by adding 1 to N such that

$$N \geq \log_2(\frac{S}{C} + 1) + 1$$

Which may be approximated such that

$$N \geq \log_2(\frac{S}{C}) + 1$$

To find the time to send S segments in N roundtrips, multiply by the RTT R:

$$T_N = (\log_2(\frac{S}{C}) + 1) \cdot R$$

... which matches the known equation for the imte needed to download and acknowledge n packets:

$$R(n) = [log_2(\frac{n}{CW_{init}}) + 1] \cdot RTT$$

Updated Total Transfer Time Formulae

$$T_{SS} = 1.5R + P + (\log_2(\frac{S}{C}) + 1)R$$

$$T_{FO} = 1.0R + P + (\log_2(\frac{S}{C}) + 1)R$$